Claims

1. (Currently amended) In a multitasking operating system that uses virtual memory to share physical memory among concurrently executing application programs, a method for controlling allocation of physical memory comprising:

in response to a ~~program~~ call from an application program, other than an operating system, ~~create a data structure~~ to group said application specified code or data in a group, creating a structure to group the code or data specified by the application;

monitoring for a not-present interrupt generated by a virtual memory system used by ~~of~~ said multitasking operating system in response to a said application request to access any part of the code or the data in the group; and

when the not-present interrupt occurs for a unit of memory in the group, loading all of the code or the data in the group that is not already in the physical memory into the ~~said~~ physical memory from secondary storage at one time, using a single series of loading operations without further not-present interrupts being generated by the ~~said~~ virtual memory system for another unit of memory in ~~said~~ the group, ~~and said~~ the loading including the unit of memory for which the not-present interrupt has occurred and all other units of memory used to store the code or the data in the group.

2. (Currently Amended) The method of claim 1 wherein the structure includes a linked list structure that links together the code or the data stored at non-contiguous portions of the virtual memory.

3. (Currently Amended) The method of claim 2 wherein the structure links pages of memory associated with the non-contiguous portions of the code or the data.

4. (Currently Amended) The method of claim 1 further including:
repeating the steps of claim 1 for additional groups of the code or the data specified by the application.

5. (Currently Amended) The method of claim 4 further including:

repeating the steps of claim 1 for a group of code or data for another concurrently executing application such that more than one concurrently executing application program has specified at

least one group of code or data to be treated as a single piece of memory for loading into the physical memory in response to [a] the not-present interrupt.

6. (Original) The method of claim 1 further including:

when the not-present interrupt occurs, checking whether the interrupt has occurred for a unit of memory in the group by evaluating whether an address of the memory request for which the interrupt occurred is within a series of non-contiguous memory addresses of the group.

7. (Currently Amended) The method of claim 1 further including:

tracking memory accesses to units of memory in the group together such that when a unit of memory in the group is accessed, all of the units of memory in the group are marked as accessed; and

determining which portions of the physical memory to swap from the physical memory to the secondary storage by determining which units of [code] memory are marked as accessed, such that the units are selected to be swapped from the physical memory to the secondary storage based on frequency of use or how recently the units of [code] memory have been accessed.

8. (Currently Amended) The method of claim 7 further including:

in response to a second call from the application program to group specified code or data in a second group, creating a second structure to group the code or data specified by the application;

tracking memory accesses to units of memory in the first and second group such that when a unit of memory in both the first and the second group is accessed, all of the units of memory in the first and the second group are marked as accessed and the unit of memory in both the first and the second group is marked as being accessed twice.

9. (Currently Amended) The method of claim 8 further including:

when a block of code or data shared between two or more groups is accessed, marking the block as being accessed n times where n is the number of groups that share the block.

10. (Currently Amended) A computer-readable medium storing instructions for performing the steps of ~~a method recited in~~ the method of claim 1.

11. (Canceled)

12. (Canceled)

13. (Canceled)

14. (Canceled)

15. (Canceled)

16. (Canceled)

17. (Canceled)

18. (Canceled)

19. (Canceled)

20. (Currently Amended) A computer-readable medium having stored thereon a data structure used for virtual memory management in a multitasking operating system, comprising:

a series of data fields forming a group for indicating blocks of code or data specified by a ~~program~~ call by an application to be treated as a single unit for purposes of virtual memory management, the data fields including a list of memory addresses of the blocks and sizes of each block in the list;

wherein the data structure is evaluated in a data processing operation to load each of the blocks into physical memory whenever a not-present interrupt is generated by ~~said~~ the virtual memory system in response to said application request for any memory address referring to a location included in one of the said blocks;

wherein the loading of the blocks into physical memory is performed at one time, using a single series of loading operations without further ~~non-present~~ not-present interrupts being generated by said virtual memory system for another unit of memory in ~~said~~ the group.

21. (Currently Amended) The computer readable medium of claim 20, wherein the list of memory addresses is an array of pointers to the blocks of memory to be placed in the group.

22. (Previously Presented) The computer readable medium of claim 20, wherein the sizes of each block in the list is indicated in an array of parameters.

23. (Currently Amended) The computer readable medium of claim 20, wherein the data structure is used to derive a linked list structure for keeping track of pages used to store the code or the data associated with the group as specified by the application.

24. (Previously Presented) The method of claim 1 further comprising, in response to a second call from the application program to further add units of memory to the group, adding the units of memory to the data structure as specified by the application.

25. (Previously Presented) The method of claim 1 further comprising, in response to a second call from the application to delete specified units of memory from the group, deleting the units of memory specified by the application from the data structure.

26. (Previously Presented) The method of claim 1 further comprising, in response to a second call from the application to destroy the group, destroying the data structure previously used for creating the group.

27. (Currently Amended) In a multitasking operating system that uses virtual memory to share physical memory among concurrently executing application programs, a virtual memory management system comprising:

means for creating a data structure to group code or data specified by one of the concurrently executing applications ~~in a group~~, in response to a ~~program~~ call from the application other than an operating system to group the specified code or data;

means for monitoring for a not-present interrupt generated by [a] the virtual memory system of ~~said~~ the multitasking operating system in response to a ~~said~~ application request to access any part of the code or the data in the group; and

means for, when the not-present interrupt occurs for a unit of memory in the group, loading all of the code or the data in the group that is not already in the physical memory into said physical memory from secondary storage at one time, using a single series of loading operations without further ~~non-present~~ not-present interrupts being generated by ~~said~~ the virtual memory system for another unit of memory in ~~said~~ the group, ~~and said~~ the loading including the unit of memory for which the not-present interrupt has occurred and all other units of memory used to store the code or the data in the group.